SSCM Exercise 3

Nikolaus Czernin - 11721138

```
library("tidyverse")
```

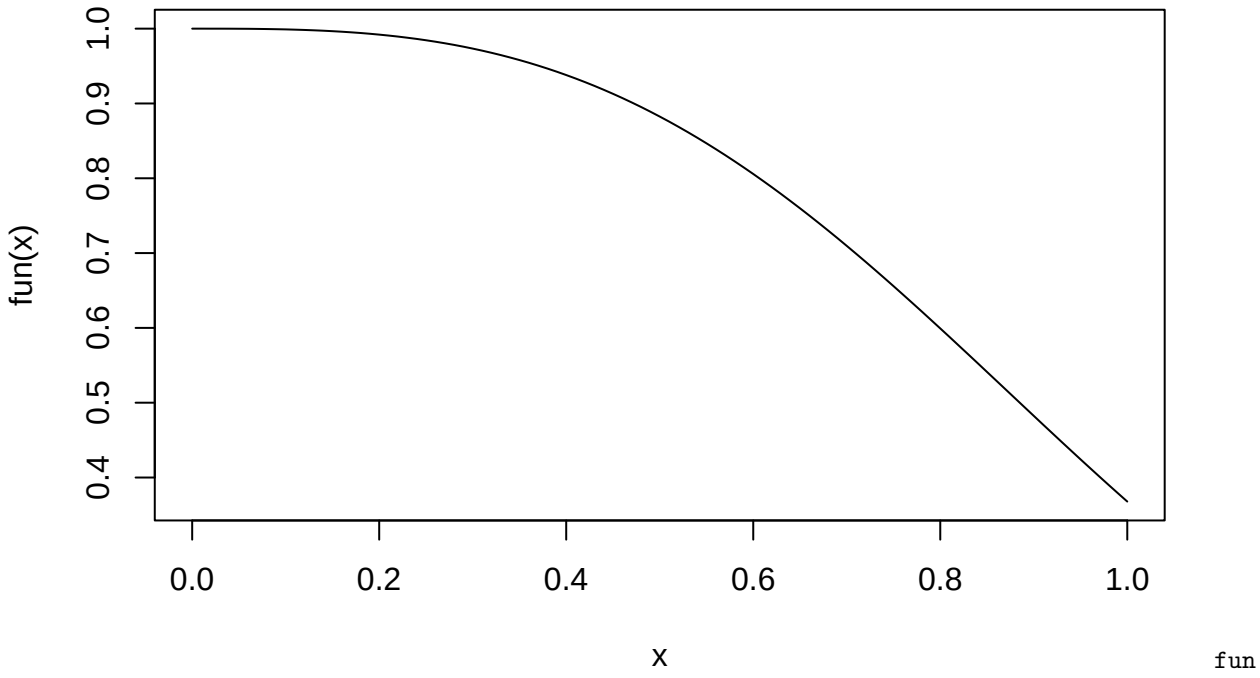
```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.2     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(knitr)
set.seed(11721138)
```

Task 1

Estimation of the integral with bounds [0, 1]

```
fun <- function(x) exp(-x^3)
curve(fun, 0, 1)
```



will be the function for the first task. I use the function `curve` to visualize it within the bounds 0 and 1.

```
us <- runif(10000) %>% sort()
mce <- mean(exp(-us^3))
int <- integrate(function(x)exp(-x^3), 0, 1)

k <- 6
print(paste("Monte Carlo Integral:", mce %>% round(k)))
```

```
## [1] "Monte Carlo Integral: 0.80822"
```

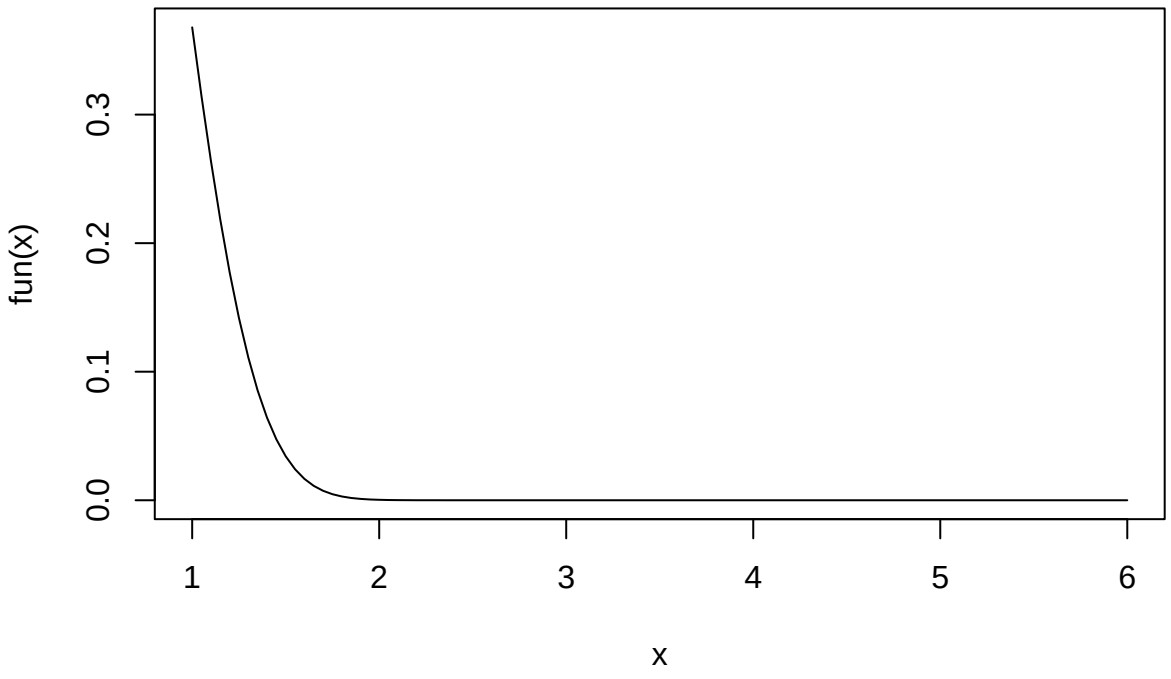
```
print(paste("Base R Integral:", int$value %>% round(k)))
```

```
## [1] "Base R Integral: 0.807511"
```

Using Monte Carlo estimation, I estimate the area under the curve by getting the mean of the functions values of 10000 unformly generated samples. Comparing the results to the result of the integrate function, we see a close match of the estimate to the actual integral.

Estimation of the integral with bounds [1, 6]

```
a = 1
b = 6
us <- runif(10000, min=a, max=b) %>% sort()
mce <- (mean(exp(-us^3))*(a-b))
int <- integrate(function(x)exp(-x^3), lower=2, upper=5)
curve(fun, a, b)
```




```
k <- 6
print(paste("Monte Carlo Integral:", mce %>% round(k)))
```

```
## [1] "Monte Carlo Integral: -0.08647"
```

```
print(paste("Base R Integral:", int$value %>% round(k)))
```

```
## [1] "Base R Integral: 2.6e-05"
```

Task 1 was to make the estimation between the bounds $[1, 6]$, an area in which the curve is already very flat and visually converging to 0. To ge tthe Monte Carlo estimate of the integral I multiply the mean of the functoin values of the sample with the difference of the bounds. Comparing the result to the output of the integrate function, we can see that the Monte Carlo integral was more precise with bounds $[0, 1]$.

Estimation of the integral with bounds [1, inf]

To estimate the integral with infinite bounds more precicely, we need to find a function with a shape similar to the function we are approximating. The exponential function is a good match because it also goes to infinity, we can therefore shift it by -1, so that the bounds of the integral are $[0, Inf)$ and we just hav eto replace x by x+1 in the function. We then divide the image of the function by the density function and get the mean to do the Monte Carlo estimation.

```
# generate samples from the exponential distribution
x <- rexp(10000)
mce <- mean(fun(x+1) / dexp(x))

int <- integrate(fun, 1, Inf)

k <- 6
print(paste("Monte Carlo Integral:", mce %>% round(k)))
```

```
## [1] "Monte Carlo Integral: 0.086923"
```

```
print(paste("Base R Integral:", int$value %>% round(k)))
```

```
## [1] "Base R Integral: 0.085468"
```

When using infinite bounds instead of 6 as an upper bound, the estimated integrals of Monte Carlo and the Base R function are much closer together. The first task used uniform sampling, which naturally grabbed a lot of samples in the region [2,6], but the density is way lower here in the target function and thus there were a lot of numbers generated which dont represent the target distribution very well and dont add enough to the sum which will be the integral.

When sampling form the exponential function we get a lot of numbers from the area of importance instead, which in turn make better contributions to the integral. There is more emphasis on smaller values, which are more commonin our target distribution.

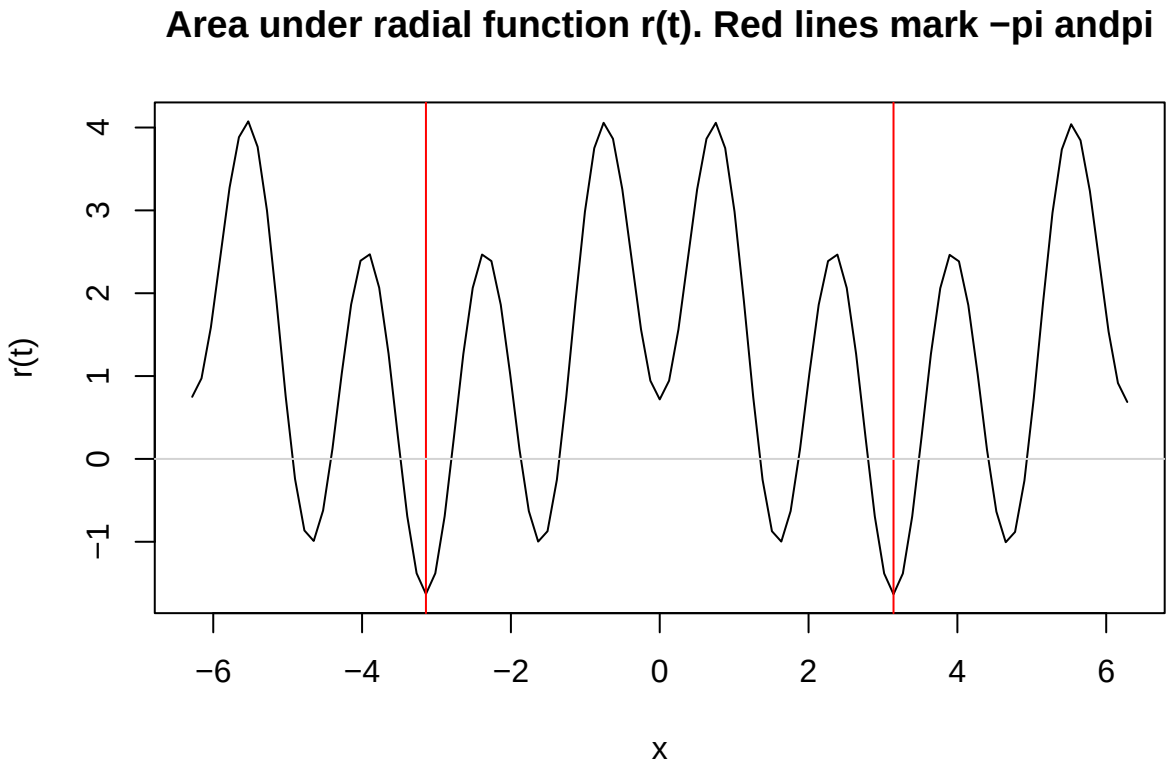
Task 2

Let $r(t) = e^{cos(t)} - 2cos(4t) - sin(\frac{t}{12})^5$

```
funr <- function(t) exp(cos(t))-2*cos(4*t) - sin(t/12)^5
```

Here I define our new function r.

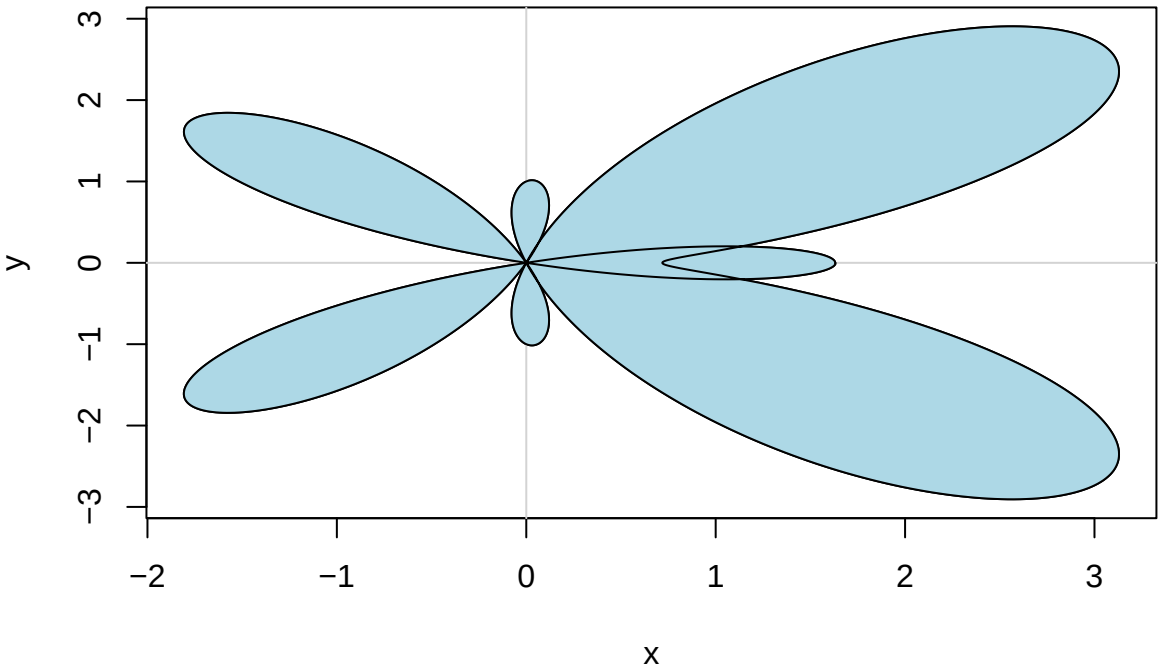
```
curve(funr, -pi*2, pi*2,
      main="Area under radial function r(t). Red lines mark -pi andpi",ylab="r(t)"
      )
abline(v=pi, col="red")
abline(v=-pi, col="red")
abline(h=0, col="lightgrey")
```




```
t <- seq(-pi, pi, length=100)
y <- funr(t)


# polygon(t, y %>% sapply(function(a) max(a, 0)), col="lightblue")
# polygon(t, -(y %>% sapply(function(a) min(a, 0))), col="lightblue")
```

Here I vsualize the radius of the function over t using the `curve` function.

```
# get cartesian coordinates
r.getx <- function(t) funr(t) * cos(t)
r.gety <- function(t) funr(t) * sin(t)

ss <- seq(-pi, pi, length=1000)
x <- r.getx(ss)
y <- r.gety(ss)

plot(x, y, type="l", main="Polar coordinates of function between -pi and pi")
abline(v=0, col="lightgrey")
abline(h=0, col="lightgrey")


polygon(x, y, col="lightblue")
```



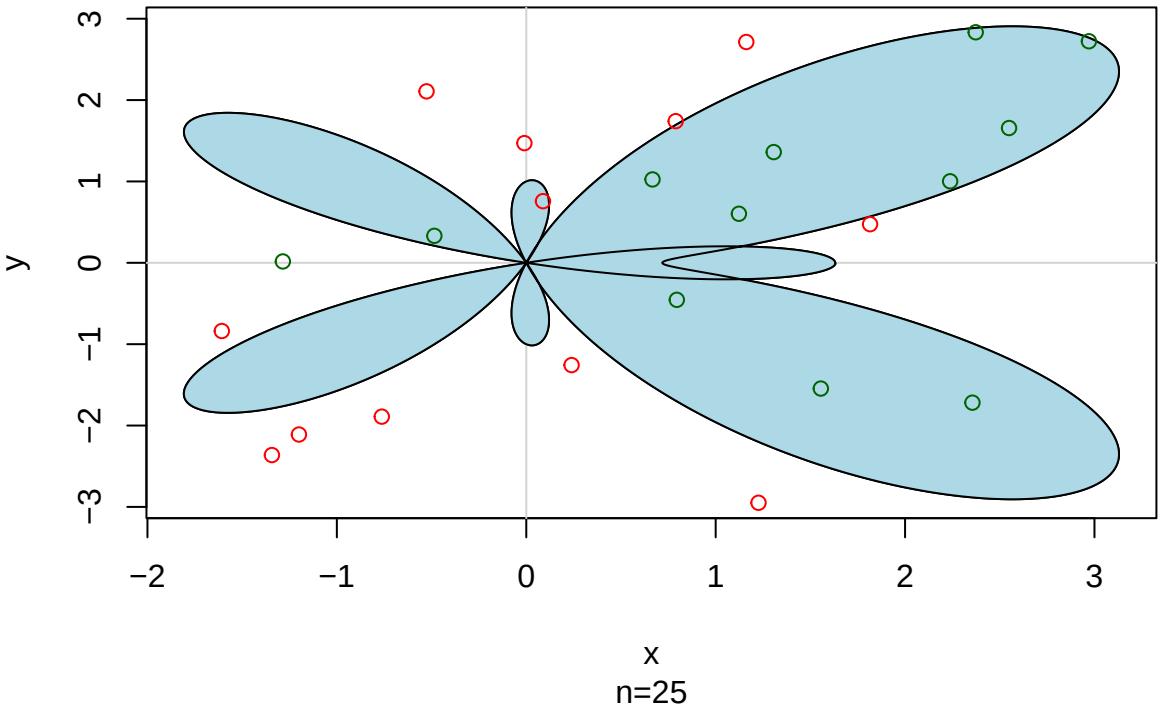
I generate evenly spaced values between -pi and pi and get their polar coordinates using the defined transformation functions. I then plot those coordinates in a cartesian coordinate system as a line and a poligon, which shows the area inside he function.

Estimating the area

```
# define a bounding box for the shape above for random numbers
# generate random coordinates in this bounding box
get_random_points_in <- function(n=1, x_min=-2, x_max=3.5, y_min=-3, y_max=3){
  # x = runif(n, x_min, x_max)
  # y = runif(n, y_min, y_max)
  # list(x=x, y=y)

  lapply(1:n, function(x)list(x=runif(1, x_min, x_max), y=runif(1, y_min, y_max)))
}
```

```
is_inside_function <- function(x, y){
  # to get the polar coordinates of x and y
  # get need radius of the point at x and y
  # and the angle
  # we know the adjacent=x and the opposite=y
  # to get the angle we need the atan2
  angle <- atan2(y, x)
  # to get the radius, we apply the pythagorean theorem
  radius <- sqrt(x^2+y^2)

  # by plugging the given angle into r() we get the radius of the
  # function in the polar coordinates
  # grab the absolute value of the radius, so that the negative
  # are under the zero-line (see radial-plot above) also counts as part
```

```
  # of the area
  actual_radius_at_angle <- funr(angle) %>% abs()

  # if the radius of the given point is smaller than the radius
  # of the function at this angle, the point is within the area
  radius <= actual_radius_at_angle
}

# test if a point at x and y is inside the function r(t)
# return a green color if yes and red color otherwise
get_point_color <- function(x, y) {
  ifelse(is_inside_function(x, y), "darkgreen", "red")
}




visualize_points_inside_r <- function(n){
  plot(x, y, type="l", main="Polar coordinates of function between -pi andpi",
       sub="n=" %>% paste0(n))
  abline(v=0, col="lightgrey")
  abline(h=0, col="lightgrey")


  polygon(x, y, col="lightblue")

  # generate n random points within the bounding box
  random_points <- get_random_points_in(n)


  # plot the points onto the graph
  for (i in 1:length(random_points)){
    p_x = random_points[[i]]$x
    p_y = random_points[[i]]$y
    points(p_x, p_y, col=get_point_color(p_x, p_y))
}
  random_points

}




ps <- visualize_points_inside_r(25)
```



To estimate the area inside this shape, we generate points within the general bounding box of the shape and test whether they are inside the shape. The plot above visualizes this problem.
I defined a function that tests, whether a point in the polar coordinate system is within the area under the curve. To do so, I need two things from the point: the angle and the radius. To get the radius, I apply the pythogorean theorem. To get the angle, I get the arctan of the ajdacent (x) and the opposite(y). I then input the angle into the function r(angle) to get the radius of the function in polar coordinates at the same angle. If then the radius of the point is smaller than the radius output by the function, the point is within the area in the shape.
I visualize this by coloring the points in the coordinate system green, if they fall into that area and red otherwise.

Here is a more crass viszalization.

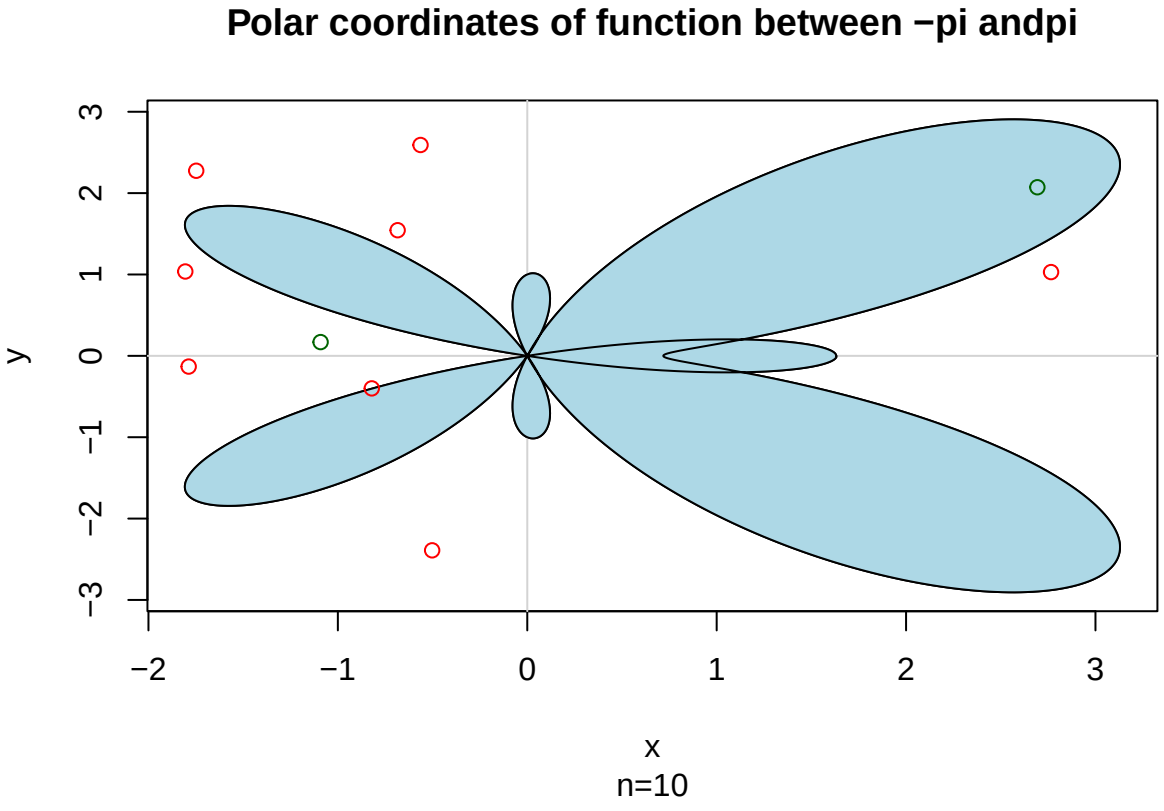
Now to estimate the area for different numbers of points

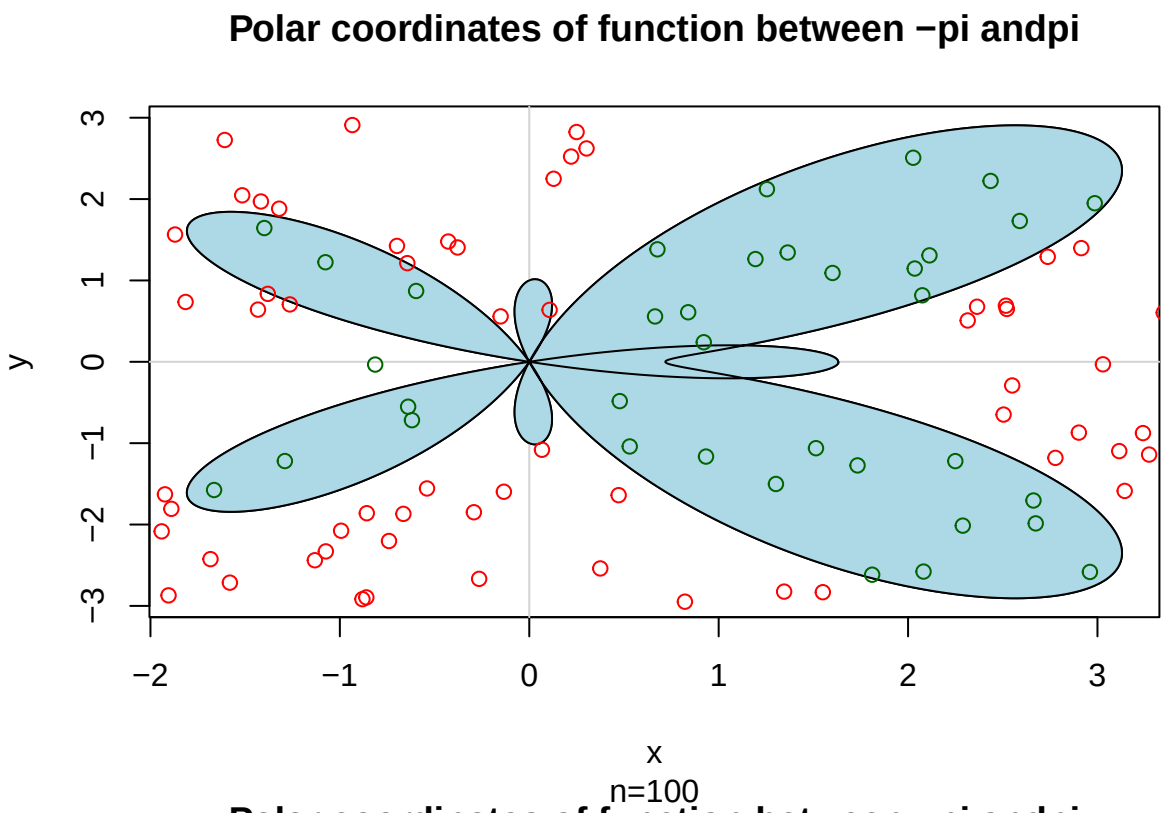
```
get_proportion_inside <- function(ps){
  # count the number of points inside the function
  n_points_inside <- sapply(ps,
                            function(p)is_inside_function(p$x, p$y)
                            ) %>%
    sum()

  proportion_inside <- n_points_inside / n
  proportion_inside
}

# the relation of the number of points inside the function to n
# is the relation of the estimated area under the curve to the total are of the bounding box
area_bounding_box <<- (3.5+2) * (3+3)
```

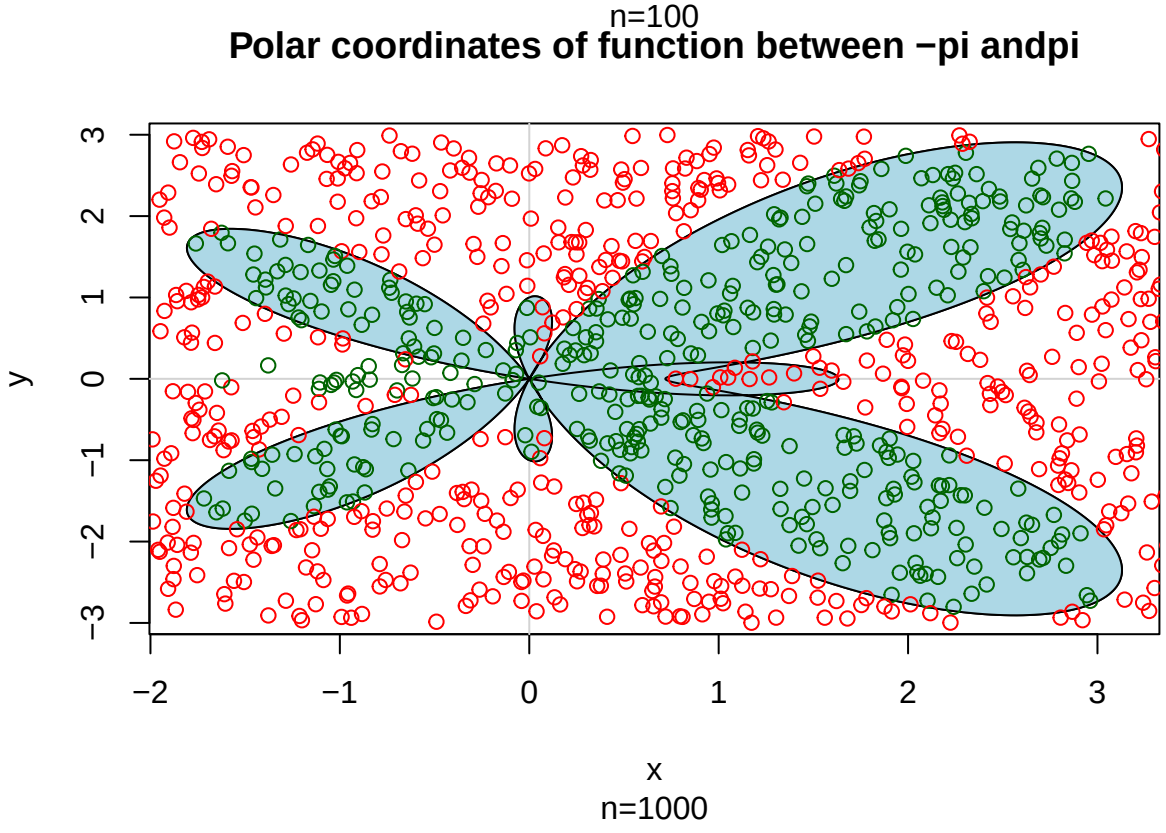
```
esimate_area <- function(prop_inside){
  area_bounding_box * prop_inside

}
```

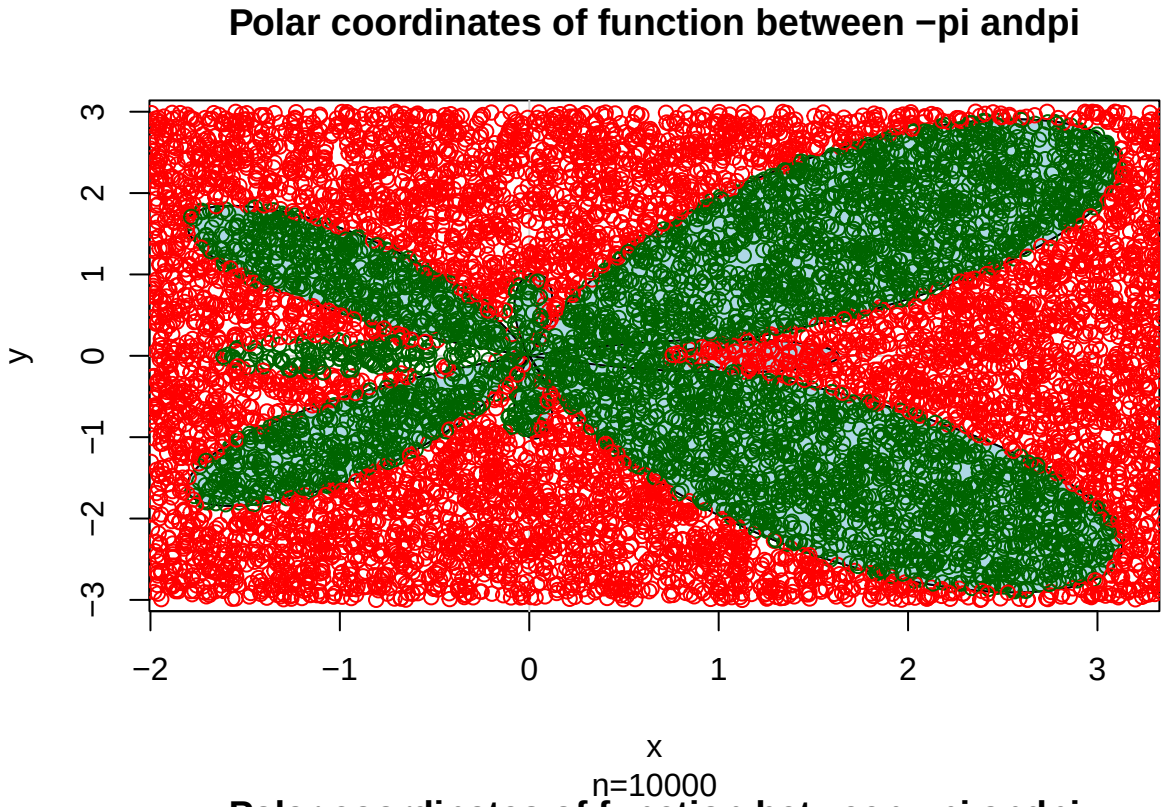
In this chunk I define a function that takes the random points given by the function defined before and returns the proportion of points inside the curv of the function. This proportion is roughly equal to the proportion of the esimtated area under the curve and the area of the whole bounding box within which the points have been generated.

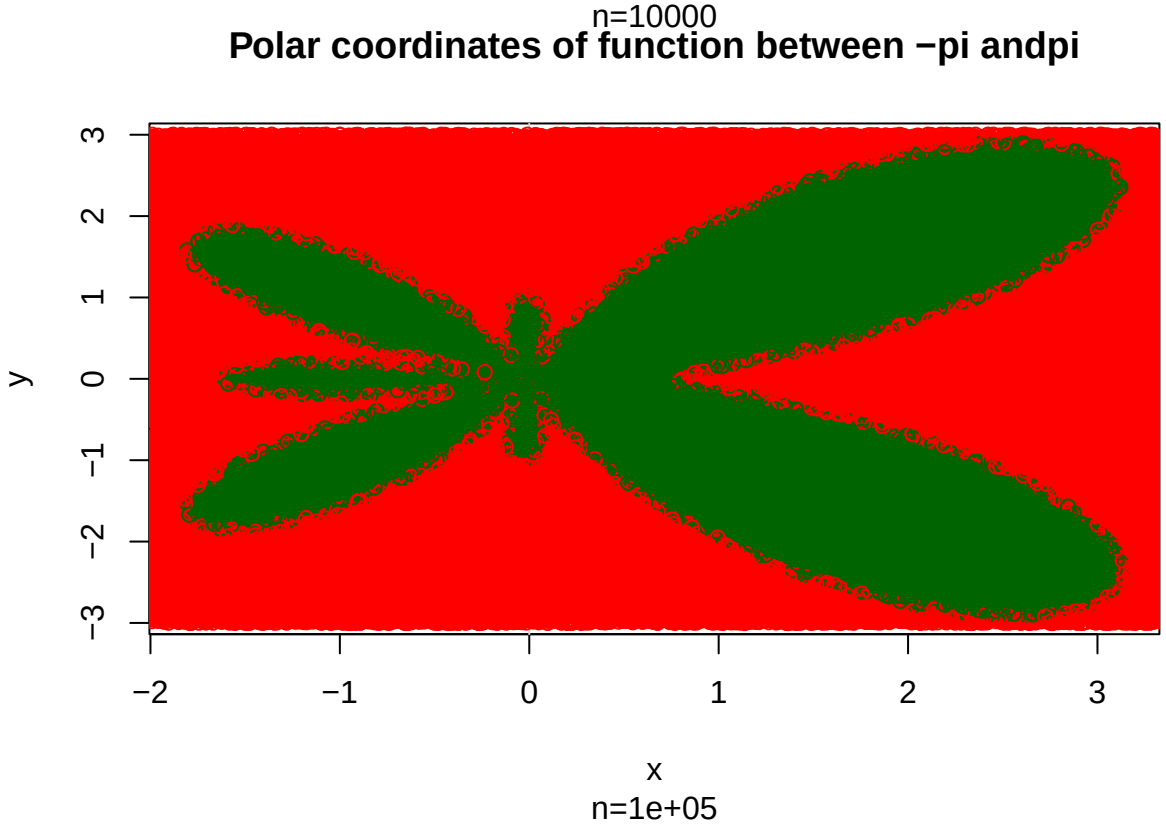
```
ns <- c(10, 100, 1000, 10000, 100000)
result <- data.frame()

for (n in ns){
  ps <- visualize_points_inside_r(n)
  prop_inside <- get_proportion_inside(ps)
  estimated_area <- esimate_area(prop_inside)
  result <- result %>% rbind(data.frame(
    n = n,
    percentage_inside_area = (prop_inside*100) %>% round(3) %>% paste("%"),
    estimated_area = estimated_area %>% round(2)
  ))
}
```




Polar coordinates of function between −pi andpi
y
x
n=100


Polar coordinates of function between −pi andpi
y
x
n=1000






```
result %>% kable()
```

n	percentage_inside_area	estimated_area
1e+01	20 %	6.60
1e+02	36 %	11.88
1e+03	43.6 %	14.39
1e+04	40.9 %	13.50
1e+05	40.789 %	13.46

The proportion of an area to is bounding box is in principe exact, when you know the area. The idea of the simulation is that the uniform distribution will yield roughly the same amount of points for each equally sized area as the number of samples grows larger. This is according to the Law of Large Numbers.
So if we generate numbers within the bounding box, the proportion of points within the section of interest to the number of points within the whole bounding box will converge to the proportion of the areas mentioned before. If we therefore generate a lot of numbers within a domain and find out how many of them are within our region of interest, we can assume that the actual area will have a similar relationship to its domain.